# Chapter 6- Pseudorandom numbers and stream cipher

**Random numbers** play an important role in the use of encryption for various network security applications, such as:

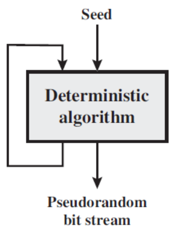
- Generating session keys,
- Generation of keys for the RSA public-key encryption algorithm
- Generation of a bit stream for symmetric stream encryption

- The following two *criteria* are used to validate that a sequence of numbers is random:

1- **Uniform distribution:** the frequency of occurrence of ones and zeros should be approximately equal.
2- **Independence:** No one subsequence in the sequence can be inferred from the others.

## Pseudorandom number generators (PRNGs)

- Cryptographic applications typically make use of algorithmic techniques for random number generation.
- These algorithms are *deterministic* and therefore produce sequences of numbers that are not statistically random.
- However, if the algorithm is good, the resulting sequences will pass many reasonable *tests* of randomness.
- Such numbers are referred to as **pseudorandom numbers**.
- a PRNG takes as input a fixed value, called the **seed**, and produces a sequence of output bits using a deterministic algorithm.
- Typically there is some feedback path by which some of the results of the algorithm are fed back as input.




## PRNG Requirements

The basic requirement is that an adversary who does not know the seed is unable to determine the pseudorandom string.

***RANDOMNESS***

In terms of randomness, the requirement for a PRNG is that the generated bit stream appear random even though it is deterministic.

- If the PRNG exhibits randomness on the basis of multiple tests, then it can be assumed to satisfy the randomness requirement.

1- **Frequency test:** The purpose of this test is to determine whether the number of ones and zeros in a sequence is approximately the same as would be expected for a truly random sequence.

2- **Runs test:** The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence.
3- **Maurer's universal statistical test:** The purpose of the test is to detect whether or not the sequence can be significantly compressed without loss of information. A significantly compressible sequence is considered to be non-random.

***UNPREDICTABILITY***

A stream of pseudorandom numbers should exhibit two forms of unpredictability:

1- **Forward unpredictability**: If the seed is unknown, the next output bit in the sequence should be unpredictable in spite of any knowledge of previous bits in the sequence.
2- **Backward unpredictability**: It should also not be feasible to determine the seed from knowledge of any generated values.

- For cryptographic applications, the seed that serves as input to the PRNG must be *secure*.
- Because the PRNG is a deterministic algorithm, if the adversary can deduce the seed, then the output can also be determined.
- Therefore, the seed must be *unpredictable*.
- In fact, the seed itself must be a random or pseudorandom number.

## Algorithm Design of PRNG

Three broad categories of cryptographic algorithms are commonly used to create PRNGs:

1- **Symmetric block ciphers.**
2- **Asymmetric ciphers**.
3- **Hash functions and message authentication codes.**

## Linear Congruential Generators

The algorithm is parameterized with four numbers, as follows:

| | | |
|---|---|---|
| $m$ | the modulus | $m > 0$ |
| $a$ | the multiplier | $0 < a < m$ |
| $c$ | the increment | $0 \le c < m$ |
| $X_0$ | the starting value, or seed | $0 \le X_0 < m$ |

The sequence of random numbers is obtained via the following iterative equation:

$$X_{n+1} = (aX_n + c) \bmod m$$

- If *a, c* and *m* are integers, then this technique will produce a sequence of integers with each integer in the range $x=0\ldots m\text{-}1$.
- We would like *m* to be very large, so that there is the potential for producing a long series of distinct random numbers.
- The generated sequence should pass the following tests:

$T_1$: The function should be a full-period generating function. That is, the function should generate all the numbers between 0 and $m$ before repeating.
$T_2$: The generated sequence should appear random.
$T_3$: The function should implement efficiently with 32-bit arithmetic.

- if is prime and c=0, then for certain values of *a* the period of the generating function is *m* - 1,
- a convenient prime value of m is $2^{31}$-1, so the generating function become:

$$X_{n+1} = (aX_n) \bmod (2^{31} - 1)$$

- some a values (such as 16807)has passed the three test.
- This generator is widely used and has been subjected to a more thorough testing than any other PRNG. It is frequently recommended for statistical and simulation works.
- The main problem with that PRNG is that its *deterministic*, which depends on the initial random value $X_0$. So, if the opponent know some parts of the sequence, he can deduce the future parts.

## Blum Blum Shub Generator

- A popular approach to generating secure pseudorandom numbers is known as the Blum, Blum, Shub (**BBS**) generator.
- The procedure is as follows.
  - First, choose two large prime numbers, and that both have a remainder of 3 when divided by 4.That is,

$$p \equiv q \equiv 3(\text{mod } 4)$$

  - For example, the prime numbers $p=7$ and $q=11$.
  - Let $n=pq$. Next, choose a random number $s$, such that $s$ is relatively prime to $n$.
  - Then the BBS generator produces a sequence of bits according to the following algorithm:

$$
\begin{aligned}
X_0 &= s^2 \bmod n \\
\textbf{for } i &= 1 \textbf{ to } \infty \\
X_i &= (X_{i-1})^2 \bmod n \\
B_i &= X_i \bmod 2
\end{aligned}
$$

*Example*: $n = 192649 = 383 * 503$ and $s = 101355$.

| $i$ | $X_i$ | $B_i$ |
|---|---|---|
| 0 | 20749 | |
| 1 | 143135 | 1 |
| 2 | 177671 | 1 |
| 3 | 97048 | 0 |
| 4 | 89992 | 0 |
| 5 | 174051 | 1 |
| 6 | 80649 | 1 |
| 7 | 45663 | 1 |
| 8 | 69442 | 0 |
| 9 | 186894 | 0 |
| 10 | 177046 | 0 |

| $i$ | $X_i$ | $B_i$ |
|---|---|---|
| 11 | 137922 | 0 |
| 12 | 123175 | 1 |
| 13 | 8630 | 0 |
| 14 | 114386 | 0 |
| 15 | 14863 | 1 |
| 16 | 133015 | 1 |
| 17 | 106065 | 1 |
| 18 | 45870 | 0 |
| 19 | 137171 | 1 |
| 20 | 48060 | 0 |

- BBS generator passes the *next-bit test*
- A pseudorandom bit generator is said to pass the next-bit test if there is not a polynomial-time algorithm that, on input of the first bits of an output sequence, can predict the bit with probability significantly greater than 1/2.
- The security of BBS is based on the difficulty of factoring $n$. That is, given $n$, we need to determine its two prime factors $p$ and $q$

### PSEUDORANDOM NUMBER GENERATION USING A BLOCK CIPHER

- A popular approach to PRNG construction is to use a symmetric block cipher (such as DES or AES) as the heart of the PRNG mechanism.
- For any block of plaintext, a symmetric block cipher produces an output block that is apparently random.
- Two approaches that use a block cipher to build a PNRG have gained widespread acceptance:
  - the CTR mode and
  - the OFB mode
- The *seed* consists of two parts:
  - the encryption key value and
  - a value that will be updated after each block of pseudorandom numbers is generated.

The CTR algorithm for PRNG can be summarized as follows.

```
while (len (temp) < requested_number_of_bits) do
      V = (V + 1) mod 2^128.
      output_block = E(Key, V)
      temp = temp || ouput_block
```

The OFB algorithm can be summarized as follows.

```
while (len (temp) < requested_number_of_bits) do
      V = E(Key, V)
      temp = temp || V
```

# STREAM CIPHERS

A typical stream cipher encrypts plaintext one byte at a time, although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time.

- a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random.
- The output of the generator, called a **keystream**, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation.

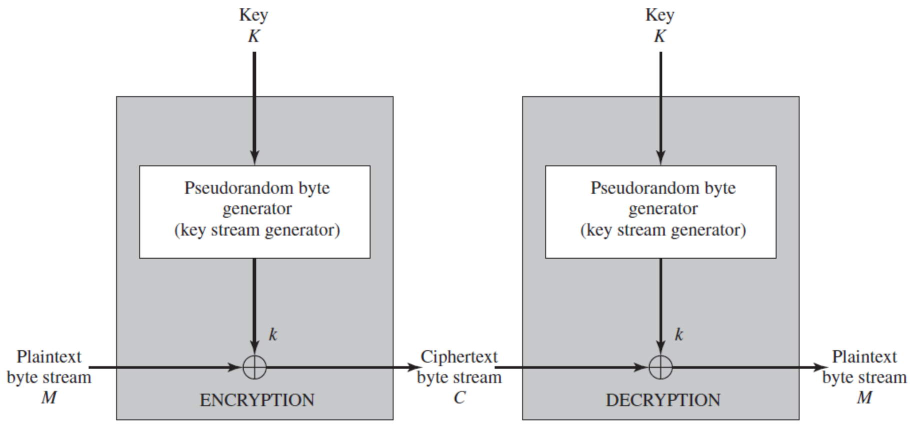



Figure 7.5 Stream Cipher Diagram

- The stream cipher is similar to the *one-time pad* discussed in Chapter 1.
- The difference is that a *one-time pad* uses a genuine random number stream, whereas a stream cipher uses a pseudorandom number stream.

## RC4

- RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security.
- It is a variable key size stream cipher with byte-oriented operations.
- The algorithm is based on the use of a random permutation.
- RC4 is used in:
    - Secure Sockets Layer/Transport Layer Security (SSL/TLS) standards
    - Wired Equivalent Privacy (WEP) protocol
    - WiFi Protected Access (WPA) protocol
- RC4 was kept as a trade secret by RSA Security, it revealed in 1994.
- A variable- length key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state vector S, with elements S[0],S[1], ؏ ,S[255].
- At all times, contains a permutation of all 8-bit numbers from 0 through 255.
- For encryption and decryption, a byte *k* is generated from S by selecting one of the 255 entries in a systematic fashion.
- As each value of is generated, the entries in S are once again permuted.

```
/* Initialization */
for i = 0 to 255 do
S[i] = i;
T[i] = K[i mod keylen];


   /* Initial Permutation of S */
   j = 0;
   for i = 0 to 255 do
       j = (j + S[i] + T[i]) mod 256;
   Swap (S[i], S[j]);
   /* Stream Generation */
   i, j = 0;
   while (true)
     i = (i + 1) mod 256;
     j = (j + S[i]) mod 256;
   Swap (S[i], S[j]);
   t = (S[i] + S[j]) mod 256;
   k = S[t];
```